Análisis Desafío 2 - Informática 2

  
 **Universidad de Antioquia**

**Desarrollado por:** Emmanuel López Ramírez

**Para el Curso de Informática 2 del Departamento de Ingeniería Electrónica y Telecomunicaciones**

En el presente se muestra el resumen exhaustivo previo al desarrollo sobre las causas y efectos de las estructuras en la implementación de una solución al Desafío 2. Por lo general es mencionable el reconocimiento de ciertas estructuras fundamentales y sus partes.

Como mención me pareció relevante el uso de una plantilla de clase para referirnos al usuario el cual representa una especie de subclases que son los anfitriones y huéspedes y con atributos genéricos con distinto valor que darán permiso o acceso a las diferentes tareas respecto a su rol.

El uso de la clase Fecha me pareció fundamental por contener una parte fundamental en la terea principal del problema y es respecto a las reservas pues su mayor limitación se encuentra en la intersección de fechas disponibles u ocupadas.

Se consideró la creación de la clase reserva al tratarse de un actor principal en el problema al tener un rol protagónico y simplificador en los métodos de las clases, pues se planea que en esta se lleven gran cantidad de las operaciones respectivas en el procesamiento con acceso o solicitud de permisos dependiendo de los atributos de otras clases con funciones amigas o simples getters. Es posible que en el desarrollo no sea viable, pero se tendrá en cuenta conforme a la marcha.

El manejo de la base de datos por el momento se mantiene en 3 archivos principales como lo son el Histórico, Vigentes y Pendientes respecto a las reservas, aún está en consideración el manejo de más TXTs.

El desarrollo de este análisis fue básico debido/gracias a la complejidad del problema, el mayor reto en la implementación será el manejo de estructuras distintas consultadas en la web que parecen cumplir con los requisitos del Desafío 2.

Se optó por un enfoque intensivo en el desarrollo de las estructuras paralelo al desarrollo con el fin de que la implementación sea lo más posible carente de confusiones o cambio de lógicas.

A continuación, se presenta la estructura de los elementos propuestos para la implementación. Me reservo el derecho de modificación en desarrollo.

Algunas tareas como la evaluación de recursos aún está en consideración.

## 1. Clases

### 2. Fecha (Valor-objeto)

Centraliza toda la lógica temporal:

- Atributos: int día, mes, año

- Operaciones:

- Constructores (incluido de copia) y destructor.

- Validación en constructor (día, mes, año válidos).

- Operadores de comparación (==, <, >, etc.).

- Métodos estáticos / amigos:

- Fecha sumarDias(int n)

- int diferenciaDias(const Fecha& otra)

### 2 Alojamiento

Representa cada oferta de “homestay”:

- Atributos:

- string código; string nombre;

- string departamento, municipio, dirección;

- double precioPorNoche;

- Lista<Amenidad> amenidades;

- Calendario calendario; // gestiona rangos ocupados

- Métodos:

- Constructor completo, copia y destructor.

- bool disponible(const Fecha& inicio, int noches) — consulta en Calendario.

- void reservar(const Fecha& inicio, int noches, const string& códigoReserva).

### 3 Reserva

Unifica activo e histórico mediante un atributo de estado:

- Atributos:

- string código; Fecha fechaEntrada; int duraciónNoches;

- string códigoAlojamiento; string documentoHuésped;

- enum Estado { Activa, Anulada, Histórica } estado;

- string métodoPago; Fecha fechaPago; double monto;

- char\* anotaciones; // buffer dinámico de hasta 1000 caracteres

- Métodos:

- Constructor, copia profunda (incluye anotaciones), destructor.

- void anular() — pone estado = Anulada.

- Serialización / deserialización (amigos de FileRepository<Reserva>).

### 4 Usuario (plantilla)

Agrupa lógica común de huésped y anfitrión sin herencia real:

template<typename TPolicy>

class Usuario {

string documento;

int antigüedadMeses;

double puntuación;

Lista<string> coleccion;

// TPolicy define si coleccion almacena alojamientos (host) o reservas (guest)

friend class FileRepository<Usuario<TPolicy>>;

};

- Policies:

- HostPolicy → colección de códigoAlojamiento.

- GuestPolicy → colección de códigoReserva.

## 2. Estructuras Auxiliares

### 1 Lista<T>

Implementación de lista simplemente ligada:

- Nodo: T dato; Nodo\* siguiente;

- Operaciones: insertar, eliminar por valor o por posición, iterador propio.

- Complejidad: O(n) en búsquedas.

### 2 Calendario

Maneja rangos de fechas reservadas en un alojamiento:

- Internamente, Lista<Rango> donde Rango { Fecha inicio; Fecha fin; }

- Métodos:

- bool encaja(const Fecha& i, const Fecha& f) — verifica no solapamiento.

- void añadir(const Fecha& i, const Fecha& f) — inserta ordenado por fecha.

## 3. Persistencia Genérica

### 1 FileRepository<T> (Plantilla)

Administra lectura/escritura sin STL, usando memoria dinámica:

template<typename T>

class FileRepository {

public:

FileRepository(const char\* rutaArchivo);

Lista<T\*> leerTodos();

void guardar(const T& obj);

void actualizar(const Lista<T\*>& objs);

private:

char\* ruta;

};

- Instanciaciones:

- FileRepository<Usuario<HostPolicy>> reposHosts("hosts.txt");

- FileRepository<Reserva> reposReservasAct("reservas\_act.txt");

- FileRepository<Reserva> reposHistorial("reservas\_hist.txt");

## 4. Capa de Orquestación

### 1 Controller

Coordina cada caso de uso sin almacenar estado permanente:

class Controller {

FileRepository<Usuario<GuestPolicy>> repoGuests;

FileRepository<Usuario<HostPolicy>> repoHosts;

FileRepository<Alojamiento> repoAlojamientos;

FileRepository<Reserva> repoReservasAct;

FileRepository<Reserva> repoHistorial;

Métricas métricas;

public:

void loginGuest(const string& doc, const string& pass);

void buscarYReservar(const Fecha&, int noches /\* filtros \*/);

void anularReserva(const string& código, bool byHost);

void consultarReservasHost(const Fecha& i, const Fecha& f);

void actualizarHistórico(const Fecha& fechaCorte);

void mostrarMétricas();

};